left in the page for more advanced handheld devices capable of receiving and displaying the images.

[0050] For the edit server, a web manager who does not want a certain web page or a web site to be compressed simply uses a standard meta tag specification to inform the server not to process selected pages from the web site or even the entire content from that web site.

[0051] **FIG. 8** illustrates an exemplary method in flowchart form for the HTML edit server of a preferred embodiment of the present invention. The editing process is performed in conjunction with the Compressor **418**, **618** and the Control Server **642** or the Redirector **416**. The process flows from a "wait state"**802** to processing once a request is received. When a request is received, the HTML Editor checks the submitted page's HTML code for tags, **804** which designate objects, such as GIF, JPEG and PNG, that can be compressed and replaced with compressed content tags. If the tag is found to be an image tag at **806** then the Compressor is notified that an image needs compression **820**. If the compression was unsuccessful the tag is skipped **824** and the Editor resumes the process of finding and examining new tags for compression **804**. If the compression was successful the image tag is edited **830**, the end of the file is reached **836**, statistics are written **838**, the file is saved **840**, the logs are updated **842** and the Editor goes back into the wait state **802**.

[0052] If the tag is found to be an Anchor Tag or a Thumbnail **808** then the editor notifies the Compressor that an image needs compression **810**. If the compression is unsuccessful the tag is skipped **814** and the editor resumes the process of finding and examining new tags for compression **804**. If the compression was successful the Anchor tag is edited, the end of the file is reached **836**, statistics are written **838**, the file is saved **840**, the logs are updated **842** and the Editor goes back into the wait state **802**.

[0053] On the other hand if the tag is a HREF that points to an image **816**, the image is downloaded **818** and the Compressor is notified that an image needs compression **820**. If the compression was unsuccessful the tag is skipped **824** and the Editor resumes the process of finding wand examining new tags for compression **804**. If the compression of the image tag was successful, the end of the file is reached, statistics are written **838**, the file is saved **840**, the logs are updated **842** and the Editor goes back into the wait state **802**.

[0054] At step **828**, the Editor checks for an image map. If an image map is found then the Editor gets the image map data **832**. If the compression is successful the Image Map is edited **830**, the end of the file is reached **836**, statistics are written **838**, the file is saved **840**, the logs are updated **842** and the Editor goes back into the wait state **802**. If an image map is not found the image tag, if any, is edited **830** the end of the file is reached **836**, statistics are written **838**, the file is saved **840**, the logs are updated **842** and the Editor goes back into the wait state **802**.

[0055] **FIG. 9** illustrates an exemplary method in flowchart form for the compression server according to an embodiment of the invention. Compression is preformed by the Compressor **418**, **618** and the HTML editor. Initially the Compressor is in a wait state until it gets a request from the HTML Editor **901**. Upon receiving a request to compress an image, the Compressor first determines the media type **902** of the material to be compressed (JPEG and GIF files are examples of media types that can be compressed). The decision block next determines whether or not the media type must first be decompressed **903**. Certain types of media such as JPEG and GIF must be decompressed before the Compressor can compress them. If there are images that need to be decompressed then decompression parameters are set **904** and the image is decompressed **905**. In the case of JPEG and GIF the decompressed image becomes a BMP (bitmap). The Compressor checks to see if the decompression was successful **906**. If the decompression was successful then the Compressor sets the compression parameters **907** and compresses the object **908**. A return code is generated **909** and the compression is checked to see if such compression was more than predetermined threshold (e.g., 10%) **910**. If the compression was more than the threshold, a reply message is sent to the Editor stating that the compression was successful **913**. The logs are then updated **914**, ending compression for this object. If the compression was less then threshold the quality is checked and a determination is made as to whether or not the quality is greater than another predetermined threshold (e.g.,>20) **911**. If the quality is >20 it is lowered by a predetermined amount (e.g., 5), **912** and the object is recompressed **908**. If the quality is less than the threshold (<20), a reply message is sent to the Editor that the compression is complete. The logs are updated **914**, thus ending the compression for this object. In the instance that the decompression is not successful **906**, a reply message is sent to the Editor stating that the decompression was unsuccessful **913**. The logs are updated **914**, thereby ending compression for this object. **FIG. 10** illustrates a flowchart of n-Depth Compression **1000** according to an embodiment of the invention. The n-Depth Compressor, working in concert with the Control Server **632** and the HTML Editor **423**, parses out links from requested URLs and sends the resulting pages to the HTML Editor. The depth of such parsing is configurable at startup. The following description of **FIG. 10** illustrates the n-Depth Compression processes. The n-depth Compressor begins in a wait state **1001** until it receives a request from the Control Server or Redirector to examine a URL for links. Upon receiving a request to examine a URL, the n-Depth Compressor locates the first link on the page **1002** and a decision block **1003** determines if the link points to a page coded with HTML. The HTML encoded page is downloaded **1004**. The depths of the associated links are calculated and the depth object is updated **1005**. The HTML is then parsed **1006** and all images are downloaded =using a separate thread **1006** and the Editor and Cache Manage are notified **1007**. Next, the Editor is notified that it needs to edit the page and then checks with the cache manager to see if it already has the page. A decision block **1008** determines whether are not the end of the page has been reached. If the end of the page has been reached, depth is recalculated and the depth object (data structure that tells the n-Depth Compressor how deep it is) is updated **1009**. If the end of the page has not been reached the routine begins again **1002** with the next link. After reaching the end of the page **1008** and recalculating depth and updating the depth object **1009**, a decision block **1010** determines whether or not n-Depth has been reached. If n-Depth has been reached the log files **1011** are updated and the n-Depth Compressor returns to wait **1001**. In the